Elmar Schömer
Ann-Christin Wörl

12. Übungsblatt

Abgabe: Dienstag, der 30.01.2024, 14:00 Uhr

Aufgabe 1: Einkaufslisten (5*+5* Punkte)

Der folgende Code findet alle `kunde_?.json` Dateien und liest diese in die Variable `entries` ein. Hierin sind Einkaufslisten von Kunden gespeichert. Diese planen in verschiedenen Supermärkten unterschiedliche Lebensmittel zu kaufen. Auf unterster Ebene sind dabei `Preis` und `Menge` der einzelnen Lebensmittel angegeben.

```
1  import glob
2  import json
3
4  files = glob.glob("kunde_*")
5  print(files)
6
7  for file in files:
8      with open(file, 'r') as f:
9          entries = json.load(f)
```

1. Implementieren Sie eine Funktion `sum_up(data)`, die ein solches verschachteltes Dictionary als Eingabe bekommt und den Komplettpreis der geplanten Einkäufe zurück gibt.
2. Zur besseren Planung möchten die Supermarktbetreiber wissen, wie viele Artikel pro Lebensmittel gekauft werden. Verwenden Sie ein Dictionary, um die Lebensmittel über alle Kunden hinweg aufzusummieren. Achten Sie darauf, die Supermärkte getrennt voneinander zu betrachten.

Aufgabe 2: Schachstellungen (10*+10*+5*+5* Punkte)

Die beiden Schachspieler Carlsen und Keymer haben sich je 10000 Schachstellungen in ihren Datenbanken gespeichert, die sie genauer analysieren wollen. Sie verwenden unterschiedliche Codierungen für eine Stellung. Beide Codierungen haben gemeinsam, dass die weißen Figuren mit den Buchstaben `K,Q,R,B,N,P` und die schwarzen Figuren mit den Buchstaben `k,q,r,b,n,p` bezeichnet werden. In der Datei `Carlsen.txt` finden Sie die Stellungen von Carlsen in der Form von 8×8-Feldern (getrennt durch /). Keymer verwendet in der Datei `Keymer.txt` die Forsyth-Edwards-Notation (siehe https://de.wikipedia.org/wiki/Forsyth-Edwards-Notation). Zur Visualisierung einer FEN kann man z.B. auf http://www.ee.unb.ca/cgi-bin/tervo/fen.pl zurückgreifen.

1. Schreiben Sie eine Klasse `Schachstellung`, die eine Schachstellung intern als ein zweidimensionales Feld von Buchstaben repräsentiert.
2. Um Objekte der Klasse `Schachstellung` in der Datenstruktur `set` (analog zu `dict`) verwalten zu können, ist es notwendig, die build-in-Memberfunktionen `__hash__` und `__eq__` zu überladen. Verwenden Sie zum Hashen einer Schachstellung den Hash-Wert eines geeigneten Strings, der diese Schachstellung in eindeutiger Weise kodiert.
3. Finden Sie heraus, welche Schachstellungen sowohl Carlsen als auch Keymer gespeichert haben. Verwenden Sie dazu die Funktion `intersection` der Klasse `set`. Lösen Sie diese Frage auch ohne Nutzung dieser Funktion.
4. Wie hoch wäre die Laufzeit einer (naiven) Vorgehensweise zur Lösung der 3. Teilaufgabe, wenn man nicht mit den Datenstrukturen `set` oder `dict` arbeiten möchte?

```
class Schachstellung:
    def __init__(self):
        self.brett = [['.' for j in range(8)] for i in range(8) ]

    def __hash__(self):
        return(hash(geeignete Zeichenkette))

    def __eq__(self,other):
        return(True / False)

C = set(Schachstellungen von Carlsen)
K = set(Schachstellungen von Keymer)

CK = C.intersection(K)
```

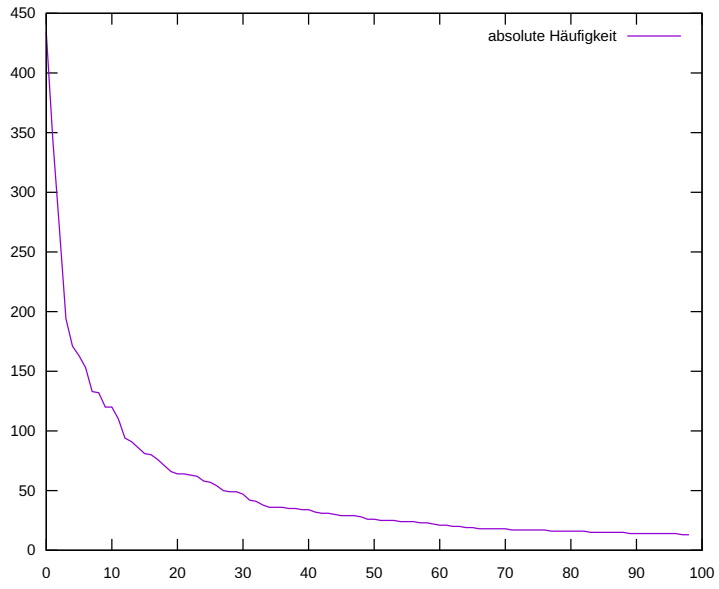
Aufgabe 3: Worthäufigkeit (5*+10*+10* Punkte)

Wir wollen die Häufigkeit von Wörtern in einem Text (z.B. "Belagerung von Mainz" von Johann Wolfgang von Goethe) untersuchen und überprüfen, ob die Häufigkeitsverteilung in etwa dem Zipfschen Gesetz (siehe https://de.wikipedia.org/wiki/Zipfsches_Gesetz) entspricht.

1. Lesen Sie die Datei `Goethe.txt` zeilenweise ein, und ignorieren Sie den Vorspann (Zeilennummer $<$ 25) und den Abspann (Zeilennummer $>$ 1430). Extrahieren Sie die einzelnen Wörter, und wandeln Sie diese mit der String-Funktion `lower` in Kleinschrift um. Achten Sie darauf, dass alle Satzzeichen wie `'.' ',' ';' '!' '"'` nicht beachtet werden.

2. Verwenden Sie ein Wörterbuch, um die Häufigkeit aller verwendeten Wörter zu ermitteln. Geben Sie die häufigsten 100 Wörter aus und beurteilen Sie die Übereinstimmung mit dem Zipfschen Gesetz.

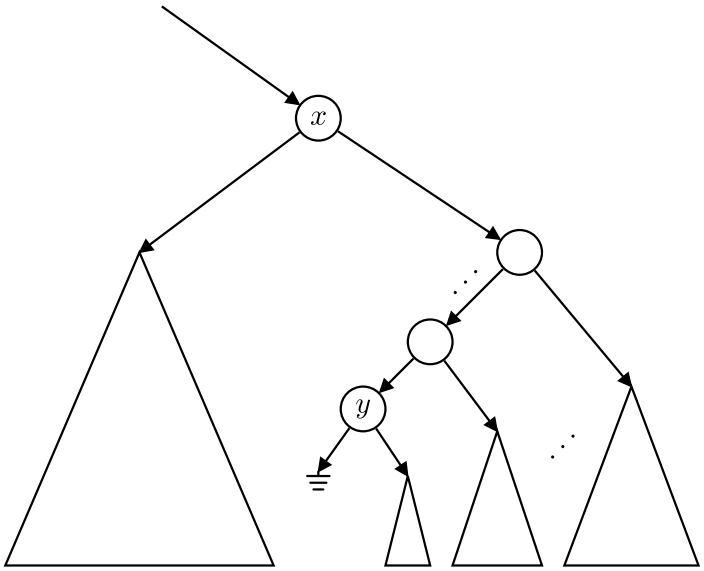
3. Wie würden Sie ohne die Verwendung eines Wörterbuchs vorgehen, um möglichst effizient die Häufigkeiten aller Wörter im Text zu ermitteln?



Aufgabe 4: Binäre Suchbäume (5*+5*+5*+5*+5*+10* Punkte)

Wir wollen die Klassen `Baumknoten` und `Suchbaum` zur Realisierung eines binären Suchbaumes modifizieren beziehungsweise erweitern.

1. Ändern Sie die Memberfunktion `suche` so ab, dass sie keine Rekursion verwendet.
2. Überladen Sie die Zugriffsoperatoren `__getitem__` und `__setitem__`, sodass folgende Benutzung eines Suchbaumes mit dem Namen `S` möglich ist: `inhalt = S[schluessel]` und `S[schluessel] = neuer_inhalt`.
3. Ergänzen Sie eine Memberfunktion `first`, die den Knoten mit dem kleinsten Schlüssel im Suchbaum ermittelt.
4. Das folgende Diagramm zeigt den allgemeinen Fall, wie man ausgehend von einem Knoten mit Schlüssel x den Knoten mit dem nächst größeren Schlüssel y ermitteln kann. Beschreiben Sie in Worten, welchen Verweisen man vom Knoten x aus folgen muss, um den Knoten y zu erreichen. Welche Spezialfälle sind zu beachten? Wie sollte man vorgehen, wenn der Knoten mit dem Schlüssel x kein rechtes Kind besitzt?



5. Wie kann man vorgehen, wenn man den Knoten mit Schlüssel x löschen möchte? Eine Erklärung in Worten genügt.

   Hinweis: Damit der Baum durch die Löschung von x nicht zerfällt, kann man den Schlüssel x des betreffenden Knotens durch den Schlüssel y ersetzen und statt dessen den Knoten mit dem Schlüssel y löschen. Begründen Sie, warum dies korrekt ist und welche Verweise man dazu abändern muss.
6. Implementieren Sie eine Memberfunktion `loesche`.